# POLYADAA: IMPROVING ALIASING REDUCTION IN MEMORYLESS NONLINEARITIES USING LAGRANGE INTERPOLATION AND POLYNOMIAL APPROXIMATION

*Leonardo Gabrielli and Stefano Squartini*

Dipartimento di Ingegneria dell'Informazione
Università Politecnica delle Marche
Ancona, Italy
l.gabrielli@staff.univpm.it

## ABSTRACT

Reducing the aliasing of nonlinear functions is an important problem in digital signal processing. The introduction of the Antiderivative Antialiasing (ADAA) method brought many benefits and is an active area of research. The current bottleneck in terms of aliasing reduction is the initial conversion from discrete- to continuous-time, which was previously done by linear interpolation. In this paper we derive PolyADAA, a method for computing the ADAA output when this conversion is done using higher order Lagrange interpolation. To obtain a viable solution, the nonlinear function is approximated using Chebyshev polynomials, which enable the ADAA integral to be computed. The paper provides numerical examples to show the effectiveness of the approach and discusses the advantages of the method.

## 1. INTRODUCTION

Nonlinear processing is an important part of digital signal processing. Static nonlinearities are employed for sound synthesis [1, 2, 3], audio effects [4], as well as emulation of physical systems [5] or virtual analog ones [6, 7].

When processing a signal with nonlinear functions, the bandwidth is generally broadened, with the risk of incurring in the generation of spurious aliasing components when signals are discrete-time. This problem is particularly important in the audio field, where those components can be perceived by the human ear even at very low levels and nonlinear processing is usually conducted in real-time with strong compute time constraints. EFficiently removing aliasing in nonlinear discrete-time processing is, thus, of paramount importance to keep a high audio quality. In the past, oversampling was the only known method to overcome the issue, until the first Antiderivative Antialiasing (ADAA) method was introduced in 2016 [8]. ADAA is conceptually based on: (a) the extraction of a continuous-time approximation of the input signal by linear interpolation; (b) the nonlinear processing; (c) convolution with an antialiasing lowpass kernel and (d) resampling at the original discrete-time instants. This work introduced the first of a family of aliasing-reduction algorithms, which spurred several new research works.

Subsequent works were mainly devoted to improving the antialiasing kernel performance by extending the finite-support kernel framework (FIR-ADAA) [9] or by introducing infinite impulse response (IIR) kernels (IIR-ADAA) [10]. Further works explored the extension of the methods to stateful systems [11, 12] and to Wave Digital Filters circuit simulation [13]. However, to the best of our knowledge, only in a recent work, [14] the initial interpolation step was addressed. Such work introduced a causal cubic interpolation strategy in the ADAA-IIR framework, leading to some improvements in the case of static nonlinearities, but with look-ahead required and without noticeable improvement with stateful nonlinearities. Another work [15] reformulated ADAA as an interpolation problem to gain some knowledge regarding its filtering effect but with no actual aliasing reduction improvement.

The objective of this work is to improve the first step of the ADAA algorithm, by employing higher order interpolation. Linear interpolation, which is currently employed in all ADAA methods except [14], is a Lagrange interpolation of order 1 [16]. We can, thus, improve that by exploiting a higher order Lagrange interpolator, however this extension is not trivial, as will be explained later and calls for additional approximation methods to obtain a closed form solution. The proposed method can be optimized and leads to a remarkable aliasing reduction.

The rest of the paper follows. Section 2 provides an overview of current ADAA methods and introduces the Lagrange interpolation formulation. Section 3 describes why extension to higher Lagrange interpolation order does not lead to a general closed form solution and proposes a Chebyshev polynomial approximation method that leads to a compact and efficient solution. In Section 4 some experiments are provided to show the validity of the method. Section 5 draws some conclusions.

## 2. BACKGROUND

ADAA can be conceptually divided in three steps to compute the output of a nonlinear waveshaping system with reduced aliasing. The first one consists in approximating the conversion from discrete-time to continuous-time using piecewise linear interpolation between the discrete-time input values $x[n]$:

$$\tilde{x}(t) = \begin{cases} x_1 + \tau(x_0 - x_1), & 0 \leq t < 1, \\ x_2 + \tau(x_1 - x_2), & 1 \leq t < 2, \\ \vdots & \\ x_n + \tau(x_{n-1} - x_n), & (n-1) \leq t < n. \end{cases} \tag{1}$$

with the auxiliary variable $\tau = 1 - (t \bmod 1)$ running in reverse with time, from 1 to 0. Then, a nonlinear function $f(t)$ is virtually applied in the con-tinuous-time domain obtaining $y(t) = f(\tilde{x}(t))$. To return to the discrete-time domain, one is left with applying

an antialiasing filter and computing the value of the output at the discrete intervals $n$. The output is, thus,

$$y[n] = \bar{y}(t) \mid_{t=nT} \tag{2}$$

where $\bar{y}(n)$ is obtained as the convolution between $y(t)$ and a lowpass filter kernel, which requires a closed-form solution of the convolution integral

$$y[n] = \int_{-\infty}^{\infty} h(u)y(n-u)\,du, \tag{3}$$

with $h$ being the antialiasing kernel.

The original paper proposed a solution based on a rectangular kernel in the range $[0, 1]$ and extended it with the convolution of two such kernels, yielding a triangular kernel in $[0, 2]$ (expressed in samples). Even though they are far from being good approximations of the ideal reconstruction filter (a $sinc$), their mild lowpass filtering proved sufficiently effective in removing aliasing and the analytical solution of the convolution integral was simple, allowing to compute the output at the discrete-time samples $y[n]$, according to Eqns. (4) or (5), respectively:

$$y[n] = \frac{F_0(x_n) - F_0(x_{n-1})}{x_n - x_{n-1}}, \tag{4}$$

$$y[n] = \frac{x_n(F_0(x_n) - F_0(x_{n-1})) - (F_1(x_n) - F_1(x_{n-1}))}{(x_n - x_{n-1})^2} + \frac{x_{n-2}(F_0(x_{n-2}) - F_0(x_{n-1})) - (F_1(x_{n-2}) - F_1(x_{n-1}))}{(x_{n-2} - x_{n-1})^2} \tag{5}$$

where the primitive functions $F_0(x) = \int f(x)dx$ and $F_1(x) = \int F_0(x)dx$ are employed.

The aliasing reduction performance can be improved by using better antialiasing filter such as higher order FIR kernels [9] or IIR filters designed in the Laplace domain [10]. However, in a subsequent work [17], the authors observed that waveshaping and wavetable synthesis can be formulated under the same light, allowing them to extend IIR-ADAA to wavetable synthesis and to note that the signal-to-noise ratio (SNR) improvement was more pronounced when the IIR filter order was increased, with respect to the results seen in [10]. The suggested explanation was that the linear interpolation was a bottleneck for performance when employed with nonlinear processing, but for wavetable synthesis - which employs linear ramps as input - this does not lead to approximation errors. This insight leads to the need for a better signal interpolation, that may go together with an improvement of the antialiasing kernel to jointly improve the SNR performance.

### 2.1. Lagrange interpolation of order $N$

The Lagrange interpolant of order $N$ is commonly written in terms of basis polynomials

$$\tilde{x}(\tau) = \sum_{k=0}^{N} x[n+k-1]\, L_k(\tau), \qquad L_k(\tau) = \prod_{\substack{j=0 \\ j\neq k}}^{N} \frac{\tau - \tau_j}{\tau_k - \tau_j}. \tag{6}$$

This form is mathematically compact and general, but it is not always the most efficient representation for numerical evaluation, especially when $\tilde{x}(\tau)$ must be evaluated many times per output sample (as in Chebyshev sampling).

For small interpolation orders ($N = 2$ or $N = 3$), it is advantageous to explicitly expand each basis polynomial $L_k(\tau)$ into monomial form,

$$L_k(\tau) = \alpha_{k,0} + \alpha_{k,1}\tau + \alpha_{k,2}\tau^2 + \cdots + \alpha_{k,N}\tau^N, \tag{7}$$

or equivalently into Horner form,

$$L_k(\tau) = \alpha_{k,0} + \tau\big(\alpha_{k,1} + \tau(\alpha_{k,2} + \cdots)\big), \tag{8}$$

where the coefficients $\alpha_{k,i}$ depend only on the interpolation nodes $\{\tau_0, \ldots, \tau_N\}$.

Once expanded, the interpolant becomes a single polynomial:

$$\tilde{x}(\tau) = b_0 + b_1\tau + b_2\tau^2 + \cdots + b_N\tau^N, \tag{9}$$

where the coefficients $b_i$ are linear combinations of the input samples:

$$b_i = \sum_{k=0}^{N} \alpha_{k,i}\, x[n+k-1].$$

This representation has several advantages:

- The polynomial can be evaluated using Horner's rule with exactly $N$ multiplications and $N$ additions per evaluation.
- All divisions are removed from the inner loop; they are absorbed into precomputed constants $\alpha_{k,i}$.
- The structure is fixed and branch-free, which is favorable for SIMD/vectorized implementations.

## 3. PROPOSED METHOD

The proposed method is based on the use of a higher order interpolation as the first step, in order to convert the signal from discrete- to continuous- time. Since linear interpolation corresponds to a Lagrange interpolation of order $N = 1$, it is conceptually easy to extend it to higher orders. However, as we will show, this does not lead to a closed-form solution. We propose, instead to approximate the function using Chebyshev polynomials. These allow to split the integral in several terms that can be computed in closed-form, provided that we use a finite-support antialiasing kernel. In the present work we will limit the discussion to the rectangular and triangular kernels, as done in [8], leaving more complex mathematical formulations of the kernel to future works.

As we will show, the Chebyshev coefficients can be computed using the Discrete Cosine Transform (DCT) making this solution particularly efficient. This yields a numerically-robust, implementable ADAA that handles quadratic/cubic Lagrange interpolation while keeping the computational cost sufficiently low.

### 3.1. Analytical Integration

A central quantity in ADAA is the integral in Eq. 3, which for a rectangular kernel is simply

$$y[n] \;=\; \int_0^1 f\big(\tilde{x}(\tau)\big)\, d\tau, \tag{10}$$

In the original ADAA formulation, we can recur to the substitution $u = \tilde{x}(\tau)$ which is constant due to the use of linear interpolation, therefore

$$\frac{d\tilde{x}}{d\tau} = x_n - x_{n-1},$$

which allows the integral (10) to be rewritten as

$$y[n] = \frac{1}{x_n - x_{n-1}} \int_{x_n}^{x_{n-1}} f(u)\, du = \frac{F(x_n) - F(x_{n-1})}{x_n - x_{n-1}}, \tag{11}$$

where $F$ is the antiderivative of $f$.

However, for higher-order Lagrange interpolation this substitution no longer works. Consider a quadratic or cubic interpolant:

$$\tilde{x}(\tau) = a_0 + a_1\,\tau + a_2\,\tau^2 \qquad \text{(order 2)}, \tag{12}$$

$$\tilde{x}(\tau) = a_0 + a_1\,\tau + a_2\,\tau^2 + a_3\,\tau^3 \qquad \text{(order 3)}. \tag{13}$$

The derivative becomes non-constant:

$$\frac{d\tilde{x}}{d\tau} = \begin{cases} a_1 + 2a_2\,\tau, & N = 2, \\ a_1 + 2a_2\,\tau + 3a_3\,\tau^2, & N = 3. \end{cases}$$

Applying the substitution $u = \tilde{x}(\tau)$ gives

$$\int_0^1 f(\tilde{x}(\tau))\, d\tau = \int_{u(0)}^{u(1)} f(u)\, \frac{1}{\frac{d\tilde{x}}{d\tau}\big(\tau(u)\big)}\, du.$$

To evaluate this integral analytically, one must invert a polynomial, which produces multiple roots. The resulting integrand has no closed form for a generic function $f$. Therefore the ADAA expression of Eq. (4) cannot be generalized. This means that for Lagrange interpolation of order $N > 1$, a different strategy is required.

### 3.2. Chebyshev expansion

Our solution consists in replacing the function $f(\tilde{x}(\tau))$ by a Chebyshev polynomial approximation on $\tau \in [0, 1]$, which admits an efficient integral representation using coefficients precomputed offline.

Remember that Chebyshev polynomials of the first kind are orthogonal functions defined as

$$T_k(x) = \cos(k \cdot \arccos(x)), x \in [-1, 1] \tag{14}$$

By introducing the map $t = 2\tau - 1$ so that the polynomial argument stays in the range $[-1, 1]$, we can approximate the nonlinear function $g(\tau) = f(\tilde{x}(\tau))$ by the Chebyshev coefficients $c_k$ as

$$g(\tau) \approx \sum_{k=0}^{K} c_k\, T_k(2\tau - 1). \tag{15}$$

with $K$ being the desired approximation order (the higher, the better).

In our setting we are interested in computing $g(\cdot)$ only at the Chebyshev nodes, that coincide with time instants

$$t_k = \cos\left(\frac{\pi(k + 1/2)}{K + 1}\right), \quad k = 0, ..., K.$$

i.e. at

$$\tau_k = \frac{t_k + 1}{2}.$$

The discrete Chebyshev coefficients can be computed beforehand as

$$c_k = \frac{2}{K} \sum_{m=0}^{K} g(\tau_m) \cos\big(\tfrac{\pi k m}{K}\big), \tag{16}$$

which can be seen as DCT-I coefficients. This is a notable property of the Chebyshev expansion allowing faster computation. The DCT-I coefficients can be evaluated by forming the symmetric vector $\mathbf{g}_{\text{sym}} = [g(\tau_0), \ldots, g(\tau_K), g(\tau_{K-1}), \ldots, g(\tau_1)]$, taking an FFT of length $2K$ and extracting the real part.

We can, thus, replace the nonlinear function $f(\tilde{x}(\tau))$ with Eq. (15) for further integration.

### 3.3. Solving the integral

In the following we want to compute the solution for either the rectangular or the triangular finite support kernels used in [8]. For now we will introduce a generic expression for the antialiasing kernel $w(\tau)$.

Consider the ADAA integral

$$y[n] = \int_0^1 f\big(\tilde{x}(\tau)\big)\, w(\tau)\, d\tau, \tag{17}$$

By substituting the Chebyshev expansion we get

$$y[n] \approx \sum_{k=0}^{K} c_k \int_0^1 T_k(2\tau - 1)\, w(\tau)\, d\tau, \tag{18}$$

where the quantity inside the integral will be called the integral moment $M_k$.

Since $t = 2\tau - 1$, then $dt = 2\, d\tau$. Let $W(t) = w(\frac{t+1}{2})/2$ so that

$$\int_0^1 w(\tau)\, d\tau = \int_{-1}^1 W(t)\, dt.$$

Then

$$M_k = \int_{-1}^1 T_k(t)\, W(t)\, dt.$$

#### *3.3.1. Rectangular kernel solution*

For the normalized rectangle $w(\tau) = 1$ on $[0, 1]$, we have $W(t) = \frac{1}{2}$ and

$$M_k^{\text{rect}} = \tfrac{1}{2} \int_{-1}^1 T_k(t)\, dt = \begin{cases} 1, & k = 0, \\ 0, & k \text{ odd}, \\ \frac{(-1)^{k/2}}{1-k^2}, & k \geq 2 \text{ even}, \end{cases}$$

#### *3.3.2. Triangular kernel solution*

For the symmetric triangular kernel, $W(t) = 2(1 - |t|)$. The moments

$$M_k^{\text{tri}} = \int_{-1}^1 T_k(t)\, 2(1 - |t|)\, dt$$

depend only on $k$ and can be precomputed numerically to high accuracy (or derived analytically for small $k$). These moments are reused for every output sample, therefore they can be precomputed offline before starting the real-time processing.

In the proposed ADAA method, the choice of antialiasing kernel affects only the Chebyshev moments $M_k$. Once these moments are precomputed, the runtime algorithm is identical for rectangular and triangular kernels. This may represent an interesting improvement with respect to Parker's method, since the kernel does not affect the algebra and hence the code, allowing fast switch from one kernel to another depending on the degree of alias reduction required (e.g. the higher the pitch of the input signal, the higher the alias reduction).

### 3.4. Algorithm overview and advantages

To summarize, the proposed method consists in an offline phase and in the actual real-time processing. The offline phase consists in computing $M_k$ integral moments after selection of the nonlinear function, the desired approximation degree (i.e. after selecting $K$), and the kernel type.

During the real-time processing the following steps must be performed:

1. Gather $N+1$ samples $x[n-1], \ldots, x[n+N-1]$.
2. Construct Lagrange basis $L_k(\tau)$ for a given fractional value $\tau$.
3. Evaluate $\tilde{x}(\tau_m)$ at Chebyshev nodes $\tau_m$, $m = 0, \ldots, K$ (Horner's rule may help here).
4. Compute $g(\tau_m) = f(\tilde{x}(\tau_m))$
5. Form symmetric vector $\mathbf{g}_{\mathrm{sym}}$ and take FFT length $2K$ to extract Chebyshev coefficients $c_k$.
6. Evaluate $y[n] = \sum_{k=0}^{K} c_k M_k$ with precomputed $M_k$.

The method has a couple of important advantages with respect to previous ADAA methods. Previous ADAA methods exhibit numerical issues when the input signal is almost stationary, because of the difference at the denominator (see Eqns. (4) and (5)). This is not the case for the current method. Furthermore, all previous methods required analytic expressions for the antiderivatives, which is removed here. Finally part of the computations are conducted offline and easy switch from triangular to rectangular kernel is possible by simply changing the precomputed Chebyshev moments.

### 3.5. Computational cost

The overall computational cost mainly depends on *(a)* the Lagrange interpolation; *(b)* the evaluation of the transcendental function $f(\tilde{x}(\tau_m))$ (this is repeated $K$ times) and *(c)* the DCT.

The Chebyshev approximation order $K$ greatly affects the computational cost, since it affects the number of transcendental function calls, the number of interpolation points $\tilde{x}(\tau_m)$ to compute and the length of the sum to finally compute $y[n]$. In most cases, computing the transcendental function will be the heaviest part, however this cost can be alleviated by the use approximations of the function that do not require transcendental functions. Although mathematical libraries such as GNU `math.h` already employ numerical tricks to make the computation as fast and precise as possible, in the context of audio DSP development, faster approximations can be devised that trade some numerical accuracy for faster compute times. However, in the context of this paper, we want to leave the discussion as general as possible, and will, thus, employ lookup tables (LUT) as a general solution to nonlinear function evaluation [18].

Computing the Lagrange interpolation of order $N$ requires, for each node $\tau_m$, to compute the linear combination of the Lagrange basis coefficients, therefore the total number of operations is approximately $(N+1) \times (1 \text{ mult} + 1 \text{ add}) \times (K+1)$ i.e. roughly $(K+1)(2N+1)$ floating-point ops.

Finally, the DCT requires a length $2K$ real FFT. The cost is approximately $K \log_2 K$. For small $K$ this can be lower than computing $K$ times the transcendental function.

Finally, the weighted sum to compute $y[n]$ requires $K+1$ multiplications + $K$ additions.

As an example, if $N = 3$ (cubic), and $K = 16$:

- Transcendental calls: $17 \times$ (cost of $\tanh$).
- Lagrange flops: $17 \times (2 \cdot 3 + 1) = 119$ flops.
- FFT of length 32

If $K$ is a power of 2, the FFT can exploit radix-2 algorithm for efficiency.

## 4. EXPERIMENTS

To validate the method, we conducted computer experiments running our Matlab implementation of the algorithm, employing a sine input and evaluating the SNR at the output of the nonlinear function, which is a standard approach for assessing aliasing reduction performance in nonlinear processing. We also implemented and tested comparative methods such as: the trivial method (no aliasing reduction), the original ADAA method using the rectangular kernel, according to Eq. (4) (from now on ADAA-R) and the original ADAA method using the triangular kernel, according to Eq. (5) (from now ADAA-T). Finally, we tested the use of LUTs to assess the speed up in compute time. In the rest of the paper, when LUTs are mentioned they employ linear interpolation. The number of points is 8192 unless stated otherwise.

In most of our experiments $f(x) = \tanh(x)$ where the input signal $x = A \cdot \sin(2\pi f_0 t)$, with amplitude $A = 9$, to drive the nonlinearity into heavy distortion. Additional experiments are reported in Section 4.2 using a hard clipping nonlinearity, keeping the same input signal and amplitude as above. In the rest of the section, where not stated otherwise, the nonlinear function is the hyperbolic tangent.

### 4.1. SNR performance

Figure 1 shows a comparison of time-domain plots of a single period of the output waveform for some of the methods. As can be seen, each method smooths the corners in its own way, with ADAA-T being the method that most suffers the processing delay.

A comparison of spectra is shown in Figure 3 for a sine tone of frequency 3192 Hz, where the spurious components are highlighted in red. As can be seen, the proposed method provides improved performance with respect to previous methods. The plot also shows how the LUT methods introduce some degree of noise floor due to spurious components introduced by the interpolated read, depending on the LUT size. However for reasonable LUT size the components are so low that they don't impair the calculated SNR value. The introduction of such components can be accepted as a trade off between compute time and quality.

To gather a more complete insight about the SNR and the various combinations of kernel type, $K$ and $N$, we ran an extensive grid of experiments, where input sine tones covering the range of the piano keyboard have been tested for SNR performance for all of the proposed methods. Specifically, tests were conducted following the proposed approach with Lagrange order $N = 2$ (quadratic) and $N = 3$ (cubic), Chebyshev moments $K = 8, 16, 32$ using both the rectangular and the triangular antialiasing kernels. The comparative methods were tested as well, to provide a clear comparison. Since the interpolated LUT read affects the SNR only by introducing a very low noise floor, we tested it only for the best combination of PolyADAA, which is triangular kernel, $K = 32, N = 3$, to see whether the result is affected by LUT read.

**Table 1:** $\overline{\Delta SNR}$ for each of the experiments, averaged among MIDI notes in the range 69-108. Please note: since all values are referred to the trivial method, its $\overline{\Delta SNR}$ would be 0 dB. LUT size for the last row of experiments is 8192 points.

| **PolyADAA-R** | | |
|---|---|---|
| | N = 2 | N = 3 |
| K = 8 | 19.33 dB | 15.99 dB |
| K = 16 | 19.34 dB | 16.16 dB |
| K = 32 | 19.34 dB | 16.16 dB |
| **PolyADAA-T** | | |
| | N = 2 | N = 3 |
| K = 8 | 10.39 dB | 24.54 dB |
| K = 16 | 10.4 dB | 27.08 dB |
| K = 32 | 10.4 dB | 27.09 dB |
| K = 32 w/ LUT | 10.38 dB | 27.08 dB |
| **Comparative methods** | | |
| ADAA-R | 5.7 dB | |
| ADAA-T | 10.6 dB | |

**Table 2:** Time elapsed by each method for processing 48k samples in frames of 64 samples. Please note that the compute time is identical for both PolyADAA-T and PolyADAA-R because the only difference lies in the moments, which are precomputed.

| **PolyADAA-R/T** | | |
|---|---|---|
| | N = 2 | N = 3 |
| K = 8 | 11.7 ms | 14.1 ms |
| K = 16 | 42.7 ms | 46.1 ms |
| K = 32 | 65.2 ms | 71.1 ms |
| K = 32 w/ LUT | 5.5 ms | 13.4 ms |
| **Comparative methods** | | |
| ADAA-R | 0.9 ms | |
| ADAA-T | 6.9 ms | |

Given the large number of experiments, and the large span of SNR values obtained along the piano key range, the most convenient way to summarize them is by referring each computed SNR to the one of the trivial method, i.e. computing the SNR difference $\Delta SNR$, which is then averaged along the piano key range to get the final value $\overline{\Delta SNR}$ seen in Table 1.

We run all these experiments to produce tones tuned to the each key of a piano keyboard (i.e. highest pitch is 4186 Hz). However, we provide results only for keys above A4 (i.e. 440 Hz or MIDI note 69), because ADAA-T would be penalized otherwise by numerical issues at low frequencies[1]. This is not seen as an issue since alias reduction is perceptually relevant only at higher pitched tones (e.g. for a sine at 440 Hz the trivial method obtains a 92.2 dB SNR). To show this effect please refer to Figure 2, where ADAA-T incurs in performances that are worse than the trivial method for notes below MIDI note 60 (i.e. $\Delta SNR < 0$). The PolyADAA-LUT method SNR is also impaired by its noise floor for low tones, e.g. with a LUT size of 8192 points the SNR for tones below A4 cannot get lower than 120 dB which is anyway perceptually irrelevant. For notes above A4 the SNR obtained by the LUT method is identical to that of the regular PolyADAA method, meaning that - above that frequency - the SNR is driven by aliasing only and not by LUT reading, making the method effective where it is most needed.

Table 1 shows the results of the tests in terms of $\overline{\Delta SNR}$. To avoid penalizing ADAA-T and PolyADAA-LUT we averaged only the SNR values computed from note A4 up to the highest piano key. The best method from our comparative is by far the more expensive PolyADAA-T (with $K = 32$, $N = 3$). However, when a computational cost trade off is needed, the choice of kernel, $K$ and $N$ is not trivial, requiring some experiments to find the most suitable combination of parameters. We can see that, e.g. the Chebyshev order - with this specific nonlinear function - can be as low as 16 to obtain the best results, given a specific kernel and Lagrange interpolation order, and most of the times $K = 8$ is sufficient. When the second order Lagrange interpolation is selected, the rectangular kernel seems to be the best choice. Looking from a different angle, increasing the Lagrange order does lower average SNR values with polyADAA-R, while it improves the SNR with polyADAA-T. The reason for this uneven behavior is that each method excels in a given frequency band, a detail that cannot be seen from the average number of the table, but can be deduced from Figure 2. The plot shows that PolyADAA with rectangular kernel, $N = 2$, $K = 8$ is the best in the test up to MIDI note 70. However, PolyADAA with triangular kernel, $N = 3$, $K = 8$, excels at keys higher than 70. These differences must be imputed to very small numerical noise, probably given by the polynomial approximation error, which has no audible effect, but impacts our SNR evaluations when the noise floor (approximately -350 dB with our double precision Matlab implementation) gets slightly higher.

Finally, we can observe that the worst of the proposed methods scores identically to the ADAA-T. This makes the method also superior to ADAA-R in terms of performance.

### 4.2. Hard-clip nonlinearity

Another frequently employed nonlinear function is the hard clip function, which can be defined as $f(x) = \frac{1}{2}(|x+1| - |x-1|)$. Differently from the hyperbolic tangent, this has discontinuities in the first derivative, making it more prone to aliasing. We conducted experiments in line with the ones above to further verify the validity of the proposed method. SNR and spectra are shown in Figure 4. As can be seen, polyADAA-R ($N = 2, K = 8$) exhibits a slightly higher SNR than the one obtained by ADAA-T, while polyADAA-T ($N = 3, K = 32$) surpasses ADAA-T by almost 20 dB.

### 4.3. Execution times and trade-offs

As a rough indication of the computational time required by the proposed method we conducted a benchmark written in plain C++ code on a Intel i7-1360P laptop running Linux. The code was written to process 1 second of signal at 48 kHz and averaged over multiple runs. The code used a block size of 64 samples. All variables

[1] Care has been taken in implementing Equations (15)-(16) from [8] and choosing the threshold that best reduced artifacts.

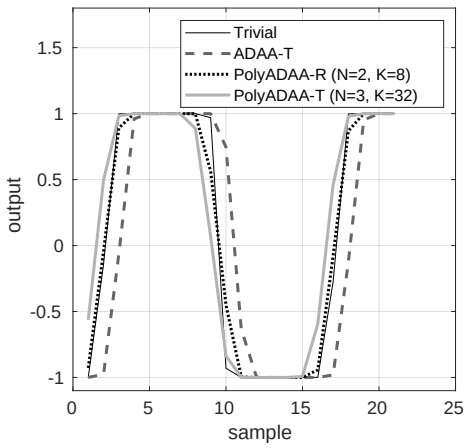



**Figure 1:** Time domain comparison of the output waveforms from the following methods: trivial (solid black), ADAA-T (dashed), PolyADAA-R ($N = 2$, $K = 8$, dotted line), PolyADAA-T ($N = 3$, $K = 32$, gray line). The same methods are evaluated in the frequency domain in Figure 3. Input sine frequency is 3192 Hz.

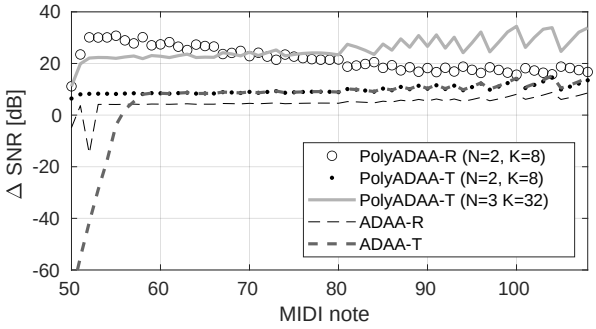



**Figure 2:** SNR improvement with respect to the trivial method. Please note that for low frequency, the triangular kernel ADAA method is subject to numerical issues that impairs its SNR. For this reason, SNR values are provided only for notes above 69 (A4 = 440 Hz).

were single-precision float and the `kiss_fft`[2] v1.0.3 library was used to compute the DCT. The nonlinearity is the same as above. All transcendental functions are computed using GCC `math.h` implementations. All runtime functions are declared `inline` to avoid context switching. Please note that the execution times of PolyADAA-R and PolyADAA-T are intrinsically identical for a given $K, N$, because they only differ in the Chebyshev moments $M_k$, which are precomputed offline.

Table 2 shows the results of the benchmark. ADAA-R and ADAA-T are faster than the proposed PolyADAA methods, as expected, since PolyADAA requires $K$ evaluations of the nonlinear function. Results may vary with different nonlinear function, therefore this benchmark must be taken only as a reference. The Chebyshev order is the parameter that most affects the execution time, as exepcted, while the Lagrange interpolation order affects it in a lesser measure. The kernel type, as outlined before, does not alter the execution time, since the Chebyshev moments are precomputed offline.

As discussed above, the nonlinear function can be evaluated using several approximation methods. One of the most general ones is the use of LUT. In these tests, the call to `std:tanh` is replaced by a linearly interpolated LUT read. This is one of the fastest method to approximate a generic nonlinear function and is accurate enough for our goals. Employing a LUT, the algorithm is noticeably fast, with vast savings between 81% and 91% for the $K = 32$ case. Therefore, by allowing a tolerable approximation error in evaluating the nonlinear function, it is possible to speed up the processing. Obviously, the same can be said about the ADAA method. The goal of this test is to give a hint of the potential optimization gains.

Finally, to evaluate a trade-off, considering the $\overline{\Delta SNR}$ results of Table 1 and the computation times of Table 2, a good SNR reduction is obtained by Poly-ADAA-R with $N = 2, K = 8$ with a reasonable compute time, but a remarkable SNR gain is obtained by PolyADAA-T with $N = 3, K = 8$ for a small execution time increase. Increasing $K$ to improve the SNR may lead to a large increase in compute time, which is probably worthless. When seeking the best SNR performance it is worth considering LUT to greatly reduce the execution time.

[2]https://github.com/mborgerding/kissfft

## 5. CONCLUSIONS

This paper addressed the initial interpolation step in ADAA to further improve the reduction of spurious components in nonlinear signal processing. All ADAA methods are currently based on linear interpolation, which cannot be easily extended to higher orders without losing the generality of the closed form solution. However, by introducing Chebyshev polynomial approximation, we were able to compute a closed-form solution that is flexible and can be solved analytically for finite support kernels such as the rectangular and the triangular ones. The method, which we called PolyADAA has advantages like the ability to effortlessly switch from rectangular to triangular kernel and the lack of numerical issue seen in previous methods. It does not require explicitly computing the antiderivatives of the nonlinear function in closed form, making it suitable to rapid adoption of new nonlinear functions without the need to analytically solve integrals. PolyADAA is quite flexible given the choice of parameters (namely, the kernel, the moments $K$ and the Lagrange interpolation order $N$).

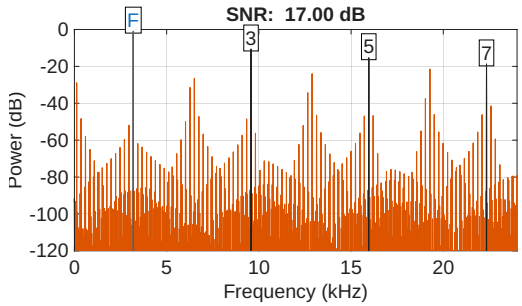
The algorithm effectively reduces aliasing, compared to ADAA-R and ADAA-T. In general, the computational cost is higher than ADAA-R or ADAA-T but the use of approximations such as lookup tables can largely reduce the cost, making it comparable to previous methods.

Overall, considering the trade-off between SNR reduction and computational cost, some combinations of the parameters offer a sweet spot of efficiency and SNR performance.

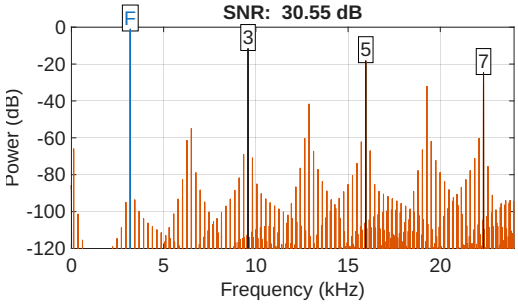
Extensions of the method to higher Lagrange interpolation order and further finite-support kernels can be devised in the future. We also believe that the proposed method could lead to vast improvements in aliasing performance if it can be coupled with IIR kernels. However, more research is required to find a suitable mathematical solution for that case.

An example Matlab implementation can be found at the com-

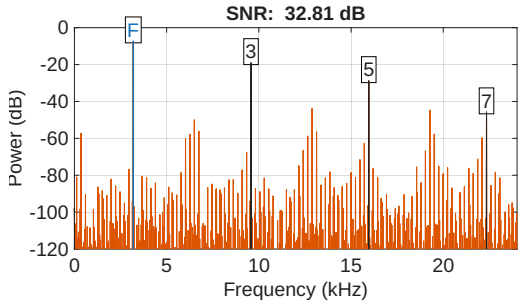
panion repository [3].



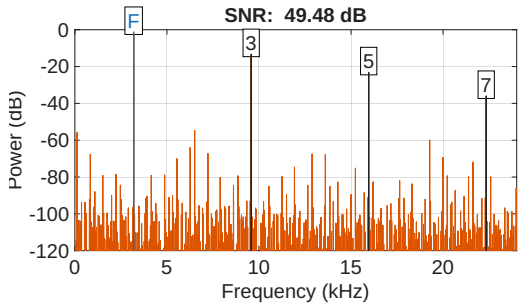

**(a)** *Trivial*




**(b)** *ADAA-T*




**(c)** *PolyADAA-R,* $N = 2$*,* $K = 8$




**(d)** *PolyADAA-T,* $N = 3$*,* $K = 32$

**Figure 4:** SNR and spectra of a sine input (3192 Hz) processed by a hard-clipping nonlinearity, using the original ADAA methods based on rectangular (a) and triangular (b) kernels, polyADAA-R and polyADAA-T with different settings (c-d). Amplitude of the sine=9, Fs=48000.

[3] `https://github.com/LOGUNIVPM/PolyADAA`

## 6. REFERENCES


[1] M. Le Brun, "Digital waveshaping synthesis," *J. Audio Eng. Soc*, vol. 27, no. 4, pp. 250–266, 1979.

[2] C. Roads, "A tutorial on non-linear distortion or waveshaping synthesis," *Computer Music Journal*, pp. 29–34, 1979.

[3] M. Puckette, *The theory and technique of electronic music*. World Scientific, 2007.

[4] P. Dutilleux and U. Zölzer, *Nonlinear Processing*. John Wiley & Sons, Ltd, 2002, ch. 5, pp. 93–136.

[5] G. Borin, G. De Poli, and D. Rocchesso, "Elimination of delay-free loops in discrete-time models of nonlinear acoustic systems," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 5, pp. 597–605, 2000.

[6] G. De Sanctis and A. Sarti, "Virtual analog modeling in the wave-digital domain," *IEEE transactions on audio, speech, and language processing*, vol. 18, no. 4, pp. 715–727, 2009.

[7] V. Välimäki, S. Bilbao, J. O. Smith, J. S. Abel, J. Pakarinen, and D. Berners, *Virtual Analog Effects*. John Wiley & Sons, Ltd, 2011, ch. 12, pp. 473–522.

[8] J. D. Parker, V. Zavalishin, and E. Le Bivic, "Reducing the aliasing of nonlinear waveshaping using continuous-time convolution," in *Proc. DAFx16, Brno, Czech Republic*, 2016, pp. 137–144.

[9] S. Bilbao, F. Esqueda, J. D. Parker, and V. Välimäki, "Antiderivative antialiasing for memoryless nonlinearities," *IEEE Signal Processing Letters*, vol. 24, no. 7, pp. 1049–1053, 2017.

[10] P. P. La Pastina, S. D'Angelo, and L. Gabrielli, "Arbitrary-order IIR antiderivative antialiasing," in *Proc. DAFx21)*. IEEE, 2021, pp. 9–16.

[11] M. Holters, "Antiderivative antialiasing for stateful systems," *Applied Sciences*, vol. 10, no. 1, p. 20, 2019.

[12] P. P. La Pastina and S. D'Angelo, "Antiderivative antialiasing with frequency compensation for stateful systems," in *Proc. Int. Conf. Digital Audio Effects*, 2022, pp. 40–47.

[13] D. Albertini, A. Bernardini, A. Sarti *et al.*, "Antiderivative antialiasing in nonlinear wave digital filters," in *Proc. DAFx2020*, 2020, pp. 62–69.

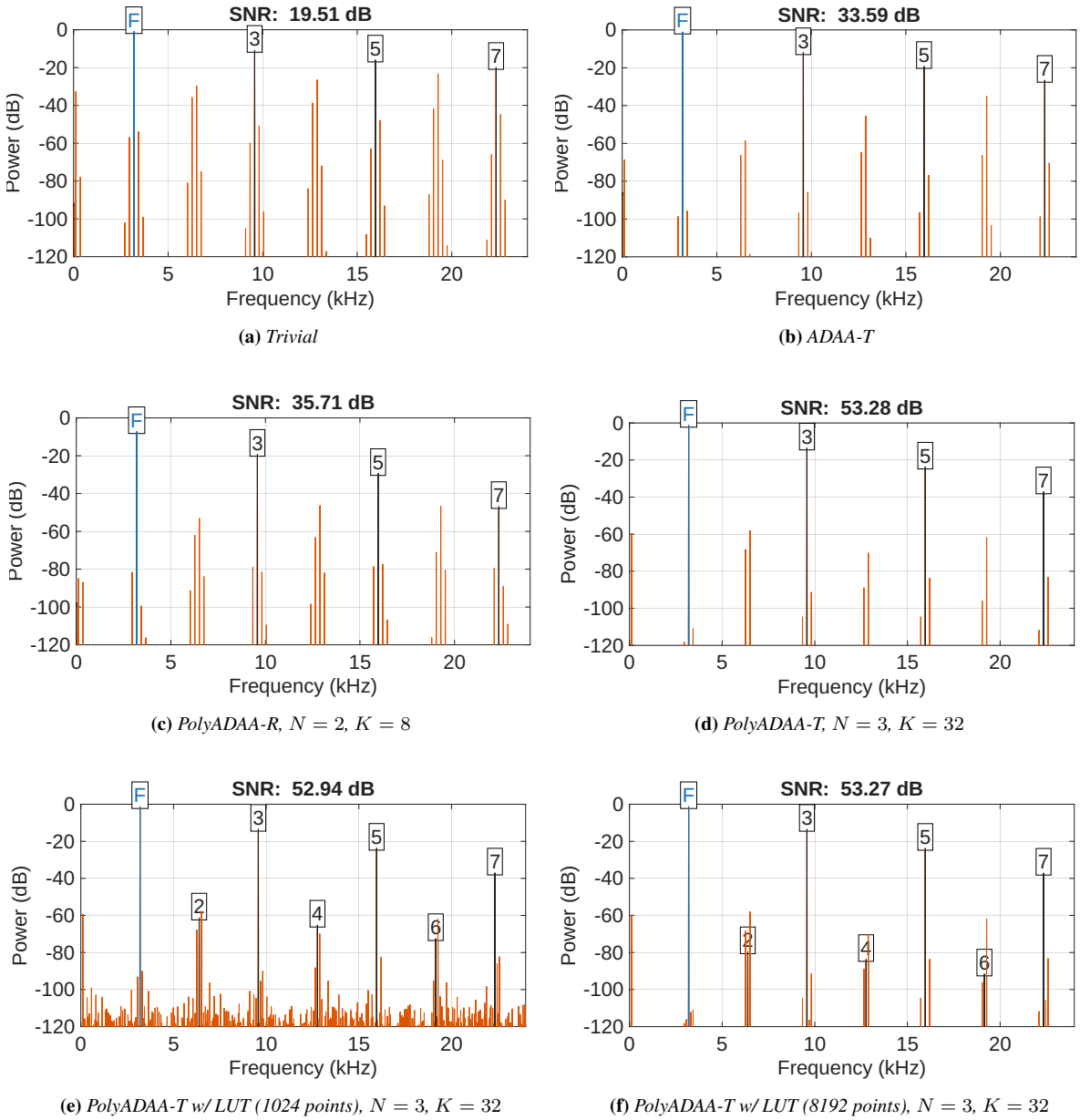
[14] V. Zheleznov and S. Bilbao, "Interpolation filters for antiderivative antialiasing," in *Proc. DAFx24*. University of Surrey, 2024, pp. 360–367.

[15] S. Bilbao, F. Esqueda, and V. Valimaki, "Antiderivative antialiasing, lagrange interpolation and spectral flatness," in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2017, pp. 141–145.

[16] V. Valimaki, "Discrete-time modeling of acoustic tubes using fractional delay filters." 1998, PhD Thesis, Aalto University, Finland.

[17] L. Gabrielli, S. D'Angelo, P. P. La Pastina, and S. Squartini, "Antiderivative antialiasing for arbitrary waveform generation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 2743–2753, 2022.

[18] L. Gabrielli, S. Squartini *et al.*, "Simplifying antiderivative antialiasing with lookup table integration," in *Proc. DAFx25*. DAFx, 2025, pp. 7–14.




**(a)** *Trivial*

**(b)** *ADAA-T*

**(c)** *PolyADAA-R*, $N = 2$, $K = 8$

**(d)** *PolyADAA-T*, $N = 3$, $K = 32$

**(e)** *PolyADAA-T w/ LUT (1024 points)*, $N = 3$, $K = 32$

**(f)** *PolyADAA-T w/ LUT (8192 points)*, $N = 3$, $K = 32$

**Figure 3:** SNR and spectra of a sine input (3192 Hz) processed by a tanh, using the original ADAA methods based on rectangular (a) and triangular (b) kernels, polyADAA-R and polyADAA-T with different settings (c-d), and polyADAA with the use of a LUT for the nonlinear function $f()$ employing either 1024 or 8192 points and linear interpolation. Amplitude of the sine=9, Fs=48000.